# Development and Application of a Heuristic to Assess Trends in API Documentation

Robert Watson
University of Washington
Seattle, WA
1-206-543-2567
rbwatson@uw.edu

## ABSTRACT

Computer technology has made amazing advances in the past few decades; however, the software documentation of today still looks strikingly similar to the software documentation used 30 years ago. If this continues into the 21st century, more and more software developers could be using 20th-century-style documentation to solve 21st-century problems with 21st-century technologies. Is 20th-century-style documentation up to the challenge? How can that be measured? This paper seeks to answer those questions by developing a heuristic to identify whether the documentation set for an application programming interface (API) contains the key elements of API reference documentation that help software developers learn an API. The resulting heuristic was tested on a collection of software documentation that was chosen to provide a diverse set of examples with which to validate the heuristic. In the course of testing the heuristic, interesting patterns in the API documentation were observed. For example, twenty-five percent of the documentation sets studied did not have any overview information, which, according to studies, is one of the most basic elements an API documentation set needs to help software developers learn to use the API. The heuristic produced by this research can be used to evaluate large sets of API documentation, track trends in API documentation, and facilitate additional research.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques—*Software libraries*

H.5.2 [Information Systems]: User Interfaces—*Training, help, and documentation; Evaluation/methodology*

## General Terms

Documentation; Human factors.

## Keywords

Application programming interface, API, API reference documentation, Software documentation, Software libraries.

## 1. INTRODUCTION

Application programming interfaces (APIs) are the interfaces through which one computer program can access the features or services provided by another program or software library. The number of these libraries and APIs is growing rapidly. Microsoft's .NET Framework is just one API of the many that Microsoft produces and it grew from 35,470 members (individual interface elements of an API) in 2002 to over 109,000 in 2007 and continues to grow [1]. If you count the work of all the other commercial independent software vendors (ISVs) and include the independent, open-source developers who also produce software libraries and APIs, the number is larger still.

At the same time, the number of software developers who use these software libraries and APIs is also growing. Whereas in the past, it was more common to write your own function rather than use one from a software library [2], the tremendous time-to-market pressures and the wide variety of software libraries available today encourage software developers to use functions provided by existing software libraries wherever possible. At the same time, today's software developers face a dual challenge of keeping up with the APIs they are familiar with as those APIs evolve and learning new APIs as they appear in the market, so they can stay current with the state-of-the-art. These conditions make it important for documentation to be effective and easy-to-use.

Thirty years ago, the situation was different—software developers did not have as many software libraries or APIs to choose from compared to the selection they have today. During the 1980s and 1990s, the personal computer boom fueled an increase in software, software libraries, APIs, and also software developers. These were the times that shaped the software documentation format that is still common today. Yet, clearly much has changed since then. Today, many more software developers are using many more software libraries and APIs under very different circumstances than existed 30 years ago. We must question, therefore, whether this documentation format from the 1980s is up to the demands of 21st-century software development. If it is, how much longer will these 20th-century formats serve the software developers of the 21st-century? More to the point, how can we find out before it is too late?

Unfortunately, it might already be too late. What few studies have been done in this area indicate that our 1980s documentation technology might no longer be up to the task and might not have been for a while. In 2002, Nykaza, et al. [3] published a study listing items that API documentation should contain. In 2009, Robillard [4] described how API users of today and tomorrow need a better way to learn about APIs by concluding: "the way to foster more efficient API learning experiences is to include more sophisticated means for developers to identify the information and the resources they need." In in their 2010 study of what makes APIs difficult to learn, Robillard and DeLine identified "inadequate API documentation as the most severe obstacle facing developers learning a new API" [5]. This observation is disappointing given that software documentation has had over 30 years to "get it right." On the other hand, such an observation invites research to find out why and what can be done about it.

This paper reviews the existing research and published literature in order to identify criteria with which to study and evaluate API reference documentation from a user-centered perspective. The resulting criteria are then used to evaluate a spectrum of API documentation from the past and present so as to test the criteria and identify challenges or gaps in their application. A method for evaluating API reference documentation at a high level in a consistent manner is the result that will enable researchers to study API reference documentation over time and ultimately improve it to meet the needs of future software developers.

## 2. BACKGROUND

The first methodological choice for studying a content set might seem to be content analysis. While content analysis would identify and measure elements of a documentation set and be able to track changes to those elements over time, it would not necessarily measure the suitability of a documentation set to a specific task or audience—that is, measure its "fit." While content analysis will be more valuable when the documentation is examined in more detail, for a high-level study, a structural analysis of user-centered requirements is more appropriate for determining the fit between the user and a product. That is, structural analysis helps one assess whether a documentation set has the basic structure required to meet the user's needs as identified in the literature.

The literature reviewed for this paper is divided into the following categories: API design and usability, online documentation design, software developers' use of documentation, and the application of heuristic evaluation.

### 2.1 API Design and Usability

References discussed here describe the fundamentals of API design and the factors used to characterize the APIs described in API reference documentation. API design refers to the design of the interface, which is related to, but separate from, the design principles of the software that actually performs a task. Consequently, these references do not necessarily describe how to implement a function with software, just how the interface a software developer uses should look and behave. While this paper focuses on API documentation, the API itself is often the first, and usually the preferred and most trusted, form of documentation [6]. As such, these design principles form the foundation of the documentation principles for APIs and they provide a means by which the design elements can be identified, measured, and compared.

Cwalina and Abrams [6] and Tulach [7] discuss the elements of an API that comprise sound design practices in object-oriented programming. Cwalina and Abrams describe design principles that are intended to provide "consistent functionality that is appropriate for a broad range of developers." They encourage a user-centered, scenario-driven approach to designing frameworks or software libraries. While Cwalina and Abrams describe their principles in the context of Microsoft's .NET Framework, many of their recommendations also apply or can be adapted to other programming environments. Tulach offers a similar design guide for software developers who create software libraries and APIs in Java, and, like Cwalina and Abrams, Tulach encourages user-centered, scenario-driven design methods.

Bloch [8] and Henning [2] discuss specifically why usability is important to consider in API design and they describe the consequences of what happens when APIs have usability problems. Bloch lists several characteristics of a good API including "Easy to learn" and "Easy to use, even without documentation" and advocates a user-centered, scenario-driven approach [8]. Henning [2] cites examples of APIs that are difficult to use successfully and easy to use in a way that reduces the program's performance and reliability. The APIs that Henning describes are designed in a way that is contrary to what Rico Mariani described as making it easy to use the API correctly and to "make it hard to do things the wrong way" [6].

Arnold [9] and Clarke [10] encourage a user-centered approach to API design by looking at the API design problem from the perspectives of human factors and usability. Arnold describes some examples of applying the user-centered design principles of understanding the audience and using progressive disclosure and Clarke describes how to characterize an API and its users along 12 different cognitive dimensions.

### 2.2 Software Developer's Documentation Use

One of the key tenets of technical writing is to know your audience and to write for them [11, 12]; however, this is complicated when the audience has diverse information-seeking goals and methods. Clarke [10] describes three groups of software developers, each with a different approach to software development and a different learning style. The three groups are opportunistic, pragmatic, and systematic [13].

Nykaza et al. [3] studied a software installation and observed how developers felt about the documentation. They listed eight ways to reduce the learning curve of an API and seven things to include in the content of a software development kit (SDK, usually a collection of software libraries, documentation, and tools necessary to include the features of third-party software into a program), and six ways to present the content. Their study, however, is now over ten years old, so some of their suggestions are now less popular, such as providing printed documentation. At the same time, studies that are more recent still arrive at many of the same conclusions.

Robillard [4] surveyed 80 software developers at Microsoft to identify the challenges they experienced learning new APIs. Robillard and DeLine [5] followed this with a more in-depth analysis to identify five dimensions of obstacles to learning an API: intent documentation, code examples, matching APIs with scenarios, penetrability, and documentation format. From their research, they derive seven implications of what software developers need in their documentation.

Rouet's TRACE model of document processing [14] is a task-oriented model of document search and information retrieval that is well suited to examining the just-in-time type of information retrieval that is frequently used by consumers of API documentation [13]. Rouet's model is a more detailed version of Wright's document interaction model [15, 16] as referenced by Nielsen: searching, understanding, and applying [17].

### 2.3 Online Documentation Design

Web content design in general has received a lot of attention; however, it is challenging to find a succinct resource that addresses the design patterns necessary to support a software developer's information-seeking needs. Redish [12] describes online document design for the user who "skims and scans." She goes on to describe some of the key design elements for this type of audience such as home pages, pathway pages, and how much content to include on a content page, elements that agree with Robillard and DeLine's [5] findings about API documentation.

Video is becoming more popular as a medium for online instruction and studies have shown that video tutorials can help users learn technical topics [18, 19, 20]. However, it is unclear how video and other new media formats can help software developers learn about an API.

## 2.4 Heuristic Evaluation of Web content

Nielsen [17] describes using heuristic evaluation as "a systematic inspection of a user interface design for usability" and a "discount usability engineering" method. The application of the heuristic studied in this paper is consistent with Nielsen's description of heuristic evaluation.

## 3. METHOD

A user-centered heuristic was derived from the published literature to study the structure of a documentation set at a high level. The resulting heuristic was tested by two coders who used it to evaluate a variety of API documentation sets.

The primary task context of the heuristic consists of the searching and understanding [17] stages of online documentation use while using API reference documentation to learn a new API. API reference documentation might have many other uses that could benefit from different heuristics, if not completely different methodologies, but they are outside the scope of this paper.

In the context of searching and understanding, readers can be divided into two groups: new readers who have not seen the site before and returning readers who have. The heuristic assumes a new reader of the documentation for several reasons. First, new readers are the more demanding of the two groups. Second, because software developers are unlikely to return to the same section of API reference documentation frequently, many characteristics of new readers apply to returning readers in this context. While it is true that returning readers might be familiar with the nature of the API when they return to the documentation, they are likely to reacquaint themselves with the site's organization if it has been a while since their last visit. In that case, the returning readers will need to search and find the information like new readers.

Additionally, the heuristic focuses only on learning to use a new API. It assumes that readers come to the documentation with a sufficient understanding of how to write a program in the context of their task and need only to learn about the features of the software library being studied.

One complication of a user-centered heuristic in this scenario is characterizing the user sufficiently. Software developers span a wide range of learning and programming styles and the software they use spans an equally diverse range of intended software development styles. The supporting literature [3, 4, 5, 12, 14], however, studied a wide range of audiences and learning styles, so the elements of the heuristic that results from this literature should also apply to a similarly diverse set of audiences.

## 3.1 Heuristic Used

The heuristic used to examine an API's documentation is based on Rouet's TRACE mode of information seeking [14], Redish's characterization of web document usage [12], and Robillard and DeLine's lists of what software developers need in API documentation [5]. Figure 1 lists the elements of the heuristic.

The heuristic has three categories: the *initial impression*, the *experience* of using the documentation, and *additional data*. Together, these categories correspond to the select document, process content, and document relevance steps in the TRACE model and "information foraging" described by Redish [12]. The factors of the initial impression describe the elements a reader would use to evaluate the nature of the site to determine if it warrants further investigation. The experience factors evaluate the elements of the documentation that affect the reader's experience after he or she decides to explore the site, and the additional data are used to collect information about the site and its evaluation.

### *3.1.1 Initial impression*

The initial impression of the documentation was evaluated by observing the following elements: the entry page content and the presence of overview information.

**Initial impression**

- Entry page content
- Overview information

**Experience**

- Top-level navigation type
- Reference topic format
- Code examples
  - Code snippets
  - Tutorials
  - Sample apps

**Additional data**

- Advanced pages
- Video tutorials
- Comments

**Figure 1. Summary of heuristic elements**

The entry page is the page seen at the top of the documentation content hierarchy. Typically, this is the page that the reader sees when he or she clicks a "documentation" link on the product's home page or it is one of the first links returned in a search for the API's documentation. This page acts as a home page for the API or the API documentation in the context of Redish [12], but it is called an *entry page* in this context to avoid confusion with the cases where it is different from the site's overall home page. The content of the entry page is a key element that a reader uses to determine the relevance of the page to his or her search and whether the site is worth exploring further. In this paper, the information elements found on the entry page were described by the coders and reviewed after all the sites had been studied.

Overview information explains the purpose or application of the API at a high level in a single page. The overview information helps characterize the *intent documentation* [5] that software developers need to orient themselves in the API. Ideally, the documentation contains an overview topic, which provides information about the scenarios for which the API was designed [3, 5]. In some cases, the entry page contains this content, while in others, this information was found on another page. In the cases where the overview information was not in the entry page, the coders used the navigational affordances of the entry page to find the overview information. If overview or intent information could not be found, this element was coded as not present.

### *3.1.2 Experience*

The experience of using the documentation was evaluated by identifying the following design elements: top-level navigation type, reference topic format, and the availability and type of code examples. The navigation style and reference-topic design elements directly characterize the documentation set's hyperlinking structure and how a reader navigates the content, which relates to how a reader maintains his or her orientation in the content [5].

The top-level navigation type used by the site describes the navigation affordances the site offers to the reader for navigating among the topics in the documentation. The sites studied used one of two different navigation styles and were coded as either hub-and-spoke or menu-content.

In the *hub-and-spoke* model, one central table-of-contents linked all other documentation. In the cases where there was no specific table of contents, this attribute was assigned if the interaction resembled such a structure, as would be the case in a breadcrumb-only navigation style, for example.

In the *menu-content* model, the pages were divided into a menu portion and a content portion—usually with the menu displayed in a narrow column the left side of the page—and the content that corresponds to the reader's menu selection displayed in a larger area to the right of the menu.

With regard to maintaining orientation in the document set, the hub-and-spoke model provides a view of the content that is limited to the current node and its sub nodes in the documentation's hierarchy, while the menu-content model generally offers a context that includes more levels of the hierarchy. For simple topics and small APIs, the view of the content offered by the hub-and-spoke model can be sufficient. In larger and more complex APIs, however, the menu-content model provides an organizational framework for the reader.

The reference topic format describes how the documentation of individual API elements is presented on the page. The API reference content could be presented as individual topics in the single-element-per-page format or as a group of elements that relate to a single object or other high-level concept in the multiple-element-per-page format.

API reference documentation usually contains a description of an element (where an element is an individual programming component provided by the software library, such as an object, class, interface, method, function, or data structure) and its parameters. These descriptions can vary in detail from simply showing the prototype or definition of the element to presenting extensively detailed descriptions of the element, including boundary cases, error returns, and other information that could be useful to the software developer. The detail an API element description requires depends on the element and its intended audience. Studying the contents of an API element description requires a more detailed examination and so they are not included in this analysis.

Code examples are an important aspect of API reference documentation [3, 5]. Robillard and DeLine [5] categorize code examples by size and complexity as code snippets, tutorials, sample applications, and production code. The first three categories can be found in published API documentation, while the fourth is usually available only to software developers who have access to the software library's source code. This looks only for code snippets, tutorials, and sample apps. For production code, the user of an open-source software library is presumed to have access to the source code behind the API, while the users of commercial software libraries are presumed not to have access to the software library's source code.

Code snippets were measured by reviewing the API reference topics to see if they included short samples of code that illustrated how that element was used. If code snippets were easily or frequently found in the documentation's topics, this value was coded as "present." If they were infrequent or could not be found in the reference topics, the value was coded as "not-present."

Tutorials, while not specifically part of API reference documentation, are important tools that help software developers understand how to use an API element or group of elements in a programming context. Software developers also use tutorials as a source from which to copy program code that they include into their own program's [5]. If tutorials could be found easily from the entry page or overview information, this value was coded as "present." If tutorials could not be found easily, the value was coded as "not present." The quality of the tutorials, such as their detail or breadth of coverage was not evaluated.

Software developers also use sample apps to understand how to use the elements of an API in context and as another source from which to copy program code [5]. Where a tutorial typically will focus on a single feature or a set of related features, a sample app is a complete application that includes examples of the API as well as the other functions that comprise a complete program, such as display, data input/output, and error handling.

### *3.1.3 Additional data*

In addition to the initial impression and experience data, the coders recorded the presence of these additional documentation elements to help characterize their experience with the documentation: advanced pages, video tutorials, and coder comments. These elements were selected to record the use of new media and technologies in API reference documentation.

Advanced pages are reference topics that allow user interaction. In contrast, a basic page uses only HTML and CSS to display content that consists of images, text, or HTML samples. Animation examples were considered basic pages unless the reader could interact with them. Advanced pages would be pages that contained elements that were more interactive or dynamic than basic pages. As an example, the *HTML5 <audio> Tag* page on the W3Schools.com site [21] would be considered an advanced page because it allows the reader to interact with the code example. Only API reference pages were considered when coding this field. Interactive tutorials or demonstration programs were not considered when coding this field.

Video tutorials were scored as "present" if video explanations about how to use the API were easily found from the entry page, overview topic, or a reference page. A video would only count if it was a narrated video clip that explained or demonstrated some aspect of the API.

The coders also recorded any Comments about the site as they reviewed the documentation.

## 3.2 API Documentation Studied

The goal was to develop a heuristic that can used to evaluate API documentation. To test the resulting heuristic, a collection of API documentation was selected for its variety in terms of API size, API source, the size of the entity that produced the API, the technology for which the API was developed, and the intended use of the API. The goal of testing the heuristic on a diverse collection of API documentation was to increase the likelihood of finding any problems with the heuristic or its application. If the heuristic proves to be valid, it could provide interesting and useful information about the sites used to test it. The intentional diversity of the APIs selected for this test also provides some insight into the current state of API documentation, even if the sample is not a statistically valid random sample that could be generalized to a larger population.

Table 1 lists the 43 software libraries that were identified to provide the desired diversity. Eight of these libraries (as indicated by an asterisk in Table 1) could not be studied for various reasons such as an inability to locate the documentation or the library did not have a conventional API. In one case, for example, a library provided a template interface and was documented more like a programming language than an API that was documented in the context of a programming language. The 35 software libraries that were studied with this heuristic supported technologies and languages such as C/C++, Java, JavaScript, PHP, Ruby, Ruby on Rails, and VAX/VMS. Of the software libraries, 29 were for a web technology, while 6 were intended for native or client software. Open-source software libraries made up 28 of the software libraries studied and 7 software libraries came from commercial independent software vendors (ISVs). The 1:4 ratio of commercial to open-source software libraries could be misleading, however. Some of the commercial libraries studied are individual libraries from commercial vendors who offer hundreds of similarly organized and authored libraries with a cumulative total of many thousands of API objects and elements. Therefore, the commercial libraries studied represent a much larger volume of API documentation topics than their numbers might otherwise indicate.

**Table 1. APIs selected to test the heuristic**

| | | | |
|---|---|---|---|
| Amazon SES | express* | knockout.js | Rails |
| ARM 4.0 | fixtext* | lettering.js* | Raphaël |
| Backbone | Google MAPS | minitest | require.js |
| batik | guard* | Mocha | rspec |
| boost | Hadoop | mustache.js* | sinon.js |
| Cake PHP | handlebars.js* | node.js | Underscore |
| CodeIgniter | Infovis | OpenVMS RTL | Microsoft Windows Documents & Printing |
| d3.js | Jasmine | paper.js | wire.js |
| Devise | Java 2 Platform Ent. Ed. V 1.4 | phantom.js | WordPress |
| Drupal | JQuery | Processing* | YUI |
| ember.js* | JQueryUI | protovis | |

* These APIs could not be studied.

The libraries sizes were measured by estimating the number of high-level objects they provided in the API. These could be objects or classes, in the case of object-oriented APIs, or functions, in the case of procedural APIs. Table 2 shows the size of the libraries used to test this heuristic.

**Table 2. Software library size**

| Size | API Objects | Total Count of Libraries Studied | Open Source Libraries |
|---|---|---|---|
| Huge | > 1000 | 4 | 1 |
| Large | 100-999 | 10 | 8 |
| Med | 10-99 | 15 | 14 |
| Small | < 10 | 6 | 5 |

## 3.3 Documentation Aspects Not Studied

The heuristic is intended to identify elements of an API documentation set along several key dimensions of utility to software developers as identified in the literature. It is designed to identify those API documentation sets that might make the APIs they document harder to learn, but this heuristic does not examine API documentation content at a detailed level. While a detailed heuristic would be a valuable tool, it is outside the scope of this paper. This heuristic identifies whether an API documentation set contains some of the aspects of API documentation that Robillard and DeLine [5], Nykaza, et al. [3], and others list as helpful to software developers learning an API, specifically, intent documentation and penetrability; however, it does not evaluate the quality of those elements.

While the usability of an API affects how a software developer learns it and what sort of documentation is required [4], this heuristic does not measure the usability or suitability of the API documentation to any particular learning style or application. The purpose of this heuristic is to study the presence or absence of documentation elements to answer the question, "does the API documentation contain the elements that help a software developer learn an API?"

## 3.4 Coding

Two experienced, professional software developers applied the heuristic to the 35 APIs that could be studied. At least one of the software developers studied each API and 15% of the results were selected at random to review for coding errors. A total of 11 coding errors were found out of 286 values coded, for an error rate of 4%. The coding errors were corrected and recoded before the data were analyzed.

## 4. FINDINGS

This section reports the observations of using the heuristic tool and of the data collected by using the heuristic.

## 4.1 Using the Heuristic

The heuristic was applied to a wide range of software libraries and API documentation styles with few coding errors. Most of the coding errors reported above resulted from having insufficiently precise operational definitions of some of the categories. For example, the "Video tutorial" category was intended to identify videos of people explaining or demonstrating a topic to the viewer. The original definition of this category, however, was worded such that it could be interpreted as any tutorial that included some video content. The documentation for a graphics library, however, included in its tutorials animated elements that demonstrated animation functions of the library. These elements were originally coded as video tutorials. However, because they did not have any presenter, they were subsequently recoded and the definition was made more specific.

Because this heuristic looks only for specific elements of a documentation set, evaluating an API documentation set is relatively quick. Coders reported spending about 10 minutes to evaluate each documentation set. The variety of documentation styles, however, provided examples that did not cleanly fit the categories described by the heuristic. For example, wiki-based documentation offered a variety of navigation styles. Some examples retained a menu, while others relied on a breadcrumb trail to show the reader where they were in the documentation. Having a breadcrumb trail alone made it difficult to maintain orientation, which is one of the problems of hypertext documentation identified by Robillard and DeLine [5].

Except for the criteria definitions that needed refinement, the heuristic seemed easy to use and apply consistently.

## 4.2 Data Collected by the Heuristic

In the course of applying the heuristic across the different sets of API documentation, an interesting view of the API documentation landscape emerged.

### *4.2.1 Document elements*

Figure 2 shows the presence of the elements that help software developers learn an API. Overview information was found in 27 of the API documentation sets evaluated and code snippets were included in the reference topics of 22 of the 35 API documentation sets evaluated. Tutorials and sample apps were less common in this collection of documentation and most sites studied used basic pages for their reference topics with only six documentation sets using advanced pages. Even fewer documentation sets used video tutorials.

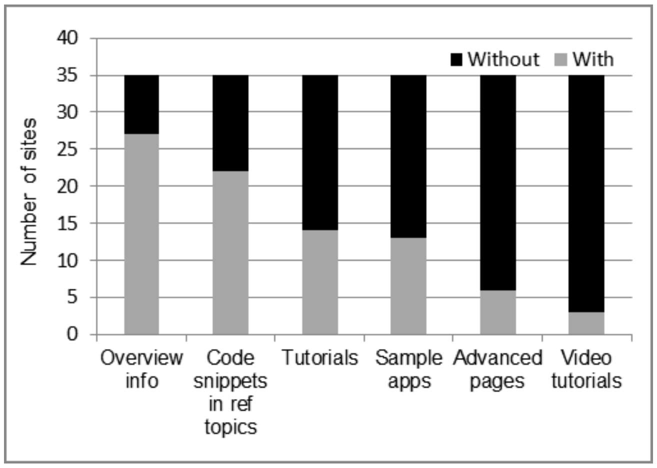

**Figure 2. Document elements found in the API documentation studied**

Figure 3 shows the elements found in the entry pages of the API documentation sites studied. The entry pages studied included such content as value propositions, which could serve as intent documentation, getting started topics, links to tutorials and the API reference, tables of contents, and in the case of a small API, the entire documentation set. Of the 35 sites studied, 12 had value propositions on their entry pages, 14 had a table-of-contents to their documentation, and 9 had getting started topics or links to getting started topics.

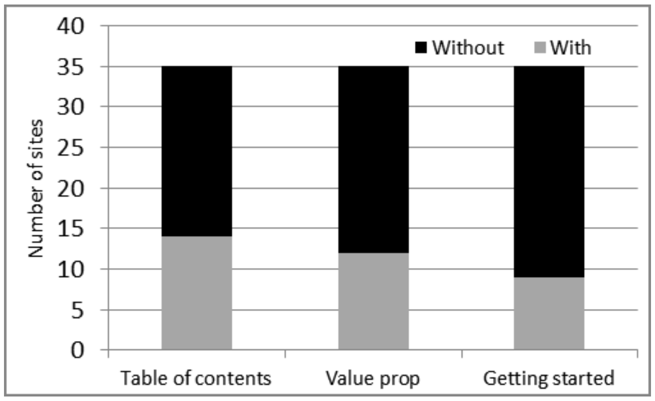

**Figure 3. Entry page contents**

### *4.2.2 Navigation elements*

The navigation model and the reference-topic page format describe the site's affordances for document navigation. Figure 4 shows that the multiple-element-per-page format was the most common format seen in the documentation studied, by a factor of about 3 to 1. The multiple-element-per-page topic model includes on a single page all the content that relates to a high-level object such as a class, interface, or object.

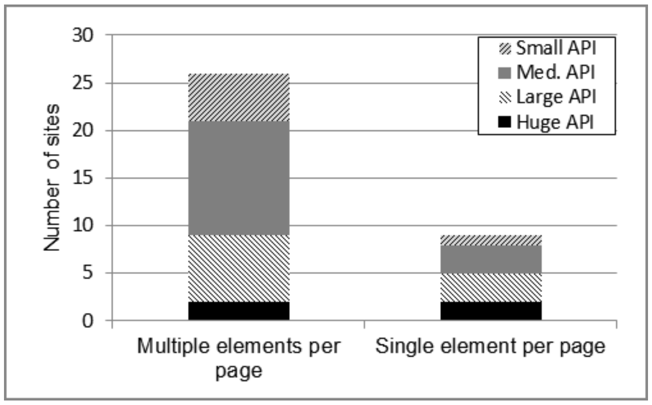

**Figure 4. Reference topic format by API size**

However, Figure 5 shows that the preference for reference-topic page format within documentation studied varies with API size. The larger APIs favored the simpler, single-topic-per-element page format, where each topic page described only one single element of an API, for example, a method, function, or structure. Smaller APIs, on the other hand were more likely to have pages in the multiple-element-per-page format, where the related elements are all on a single page.

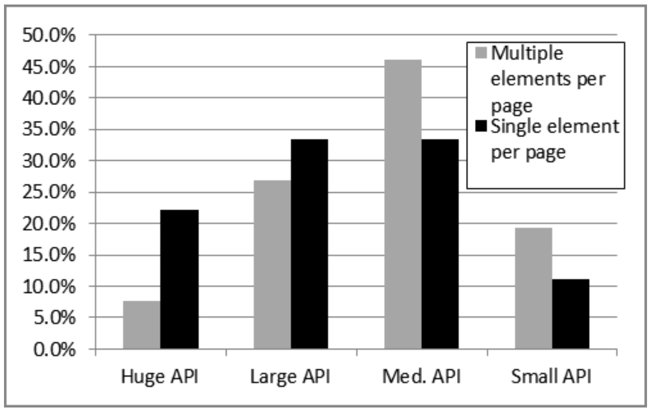

**Figure 5. Reference topic format distribution by API size**

Figure 6 shows that the menu-content style of navigation was twice as popular as the hub-and-spoke style in this collection. However, Figure 7 shows that unlike the reference topic format, the navigation model preference did not change significantly with the size of the API.

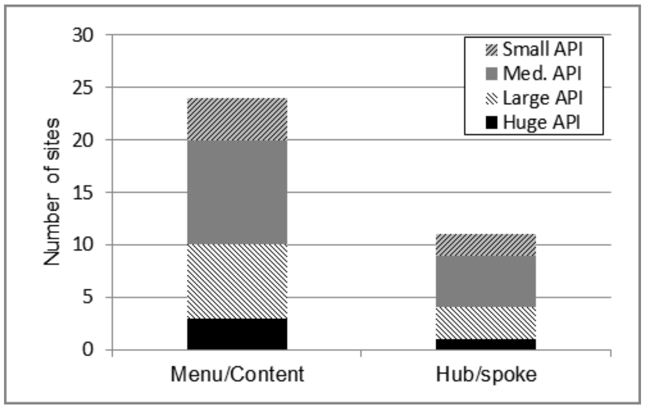

**Figure 6. Navigation model by API size**

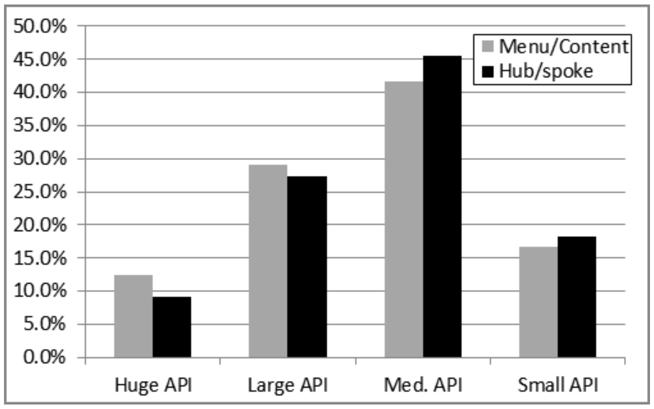

**Figure 7. Navigation model distribution by API size**

## 5. DISCUSSION

The goal of this project was to develop and test a heuristic for evaluating API documentation. The heuristic developed seemed easy to apply and accurately describes a documentation set in terms of having the high-level documentation elements that software developers need to learn an API. Because the heuristic tests for only the high-level elements of a documentation set, it is more useful to characterize a collection of API documentation than a single documentation set. Using this heuristic to survey a collection of APIs, such as those described in this paper, provides a high-level view of how that collection of documentation addresses the documentation needs commonly expressed by software developers [5].

Studying this collection of API documentation provides some interesting insights in API documentation that might warrant further investigation. For example, while video tutorials are useful and effective in teaching technical topics, very few of the API documentation sets studied included them. Somewhat more common than videos, but still found in less than half of the sites studied are the elements most desired by developers in Robillard and DeLine's [5] study: tutorials and sample apps.

Code snippets and overview information were the most common documentation elements found in the sites studied; however, eight, almost 25% of the sites studied, did not have any overview information. This is surprising considering that overview information is some of the easiest content to write and offers the much needed orientation and intent information for the API. In one example without an overview, the documentation appeared to assume the reader understood the overview and intent by going right into applications and examples. In another, it appeared that the overview was divided into sub-overviews that did not seem to relate directly to the API documentation reviewed. This particular API was huge so it is possible that the overview was elsewhere in the documentation. However, even if that was the case, it still seems risky to assume the reader will always know how to find the overview content without an affordance in the reference content.

Additional aspects of API documentation were also encountered while testing the heuristic. For example, some API documentation could not be found by online searches due to name collisions. One API had a name that was descriptive to a fault. Its name described its function by using a term that matched a common industry term. As a result, the search results contained mostly references to the term in its industry context, not the API's. As a result, the documentation was not found and so it could not be included in the APIs studied. Another aspect that was not coded by this heuristic was the usability of the site from the perspective of the coder. In the comments, some sites were noted as being easy to navigate while others were extremely difficult. These observations might be worthy to model and note in a future revision of this heuristic, but they were not modeled in this heuristic.

## 6. CONCLUSIONS AND FUTURE WORK

Good API documentation begins with the API and extends into the online documentation [6, 8]. Heuristics have been applied to measure the usability of an API [10] but not to the API's documentation. This heuristic is a step towards being able to measure the usability and suitability of API documentation sets for software developers and their tasks. It can also be used to study large collections of APIs over time to track trends.

Some of the directions that future research could take include going broader and deeper. Going broader, the heuristic could be applied to larger or targeted sets of APIs to generalize the nature of those APIs. Going deeper, new heuristics could be developed to study the documentation elements in detail to evaluate their usability and effectiveness. Other measures to evaluate include how easy the site is to find through search, how easy the site is to navigate, and the user's satisfaction with the site.

While the literature reviewed describes what software developers would like to see in API documentation, it is unclear that having those elements actually improves a software developer's learning experience, satisfaction, or ability to complete a programming task. Starting with the elements identified by this heuristic, experiments could be designed to test an API's effectiveness in specific scenarios. For example, two types of reference topic formats are described in this paper: multiple-element per page and a single-element per page. While the multiple-element per page format was more common, it is unclear whether what format makes it easier for a software developer to learn an API. With specific elements and scenarios defined, experiments could be designed to test the performance of the different elements in those scenarios.

## 7. ACKNOWLEDGEMENTS

I would like to thank Dr. Jan Spyridakis and her Internet-Based User-Experience Lab in the University of Washington's Department of Human Centered Design & Engineering for their help in collect and coding this data.

## 8. REFERENCES

[1] Abrams, B. 2008. *Number of Types in the .NET Framework*. 17 March 2008. [Online]. Available: http://blogs.msdn.com/b/brada/archive/2008/03/17/number-of-types-in-the-net-framework.aspx. [Accessed 23 January 2010].

[2] Henning, M. 2007. API design matters. *ACM Queue*. pp. 24-36, May/June 2007.

[3] Nykaza, J., Messinger, R., Boehme, F., Norman, C., Mace, M. and Gordon, M. 2002. What programmers really want: Results of a needs assessment for SDK documentation. In *Proceedings of the 20th Annual International Conference on Computer Documentation* (Toronto, Ontario, Canada, 2002).

[4] Robillard, M. P. 2009. What makes APIs hard to learn? Answers from developers. *Software, IEEE*. vol. 26, no. 6, pp. 27-34 (November/December 2009).

[5] Robillard, M. P. and DeLine, R. 2011. A field study of API learning obstacles. *Empirical Software Engineering*. pp. 703-732 (2011).

[6] Cwalina K. and Abrams, B. 2009. *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries*. Indianapolis, IN: Addison Wesley.

[7] Tulach, J. 2008. *How to Design a (Module) API.* [Online]. Available: http://openide.netbeans.org/tutorial/apidesign.html. [Accessed 26 April 2008].

[8] Bloch, J. 2005. *How to Design a Good API and why it Matters*. 19 October 2005. Available: http://lcsd05.cs.tamu.edu/slides/keynote.pdf [Accessed 27 January 2010].

[9] Arnold, K. 2005. Programmers are people, too. *ACM Queue*, pp. 55-59 (June 2005).

[10] Clarke, S. 2004. Measuring API usability. *Dr. Dobbs Journal Special Windows/.NET Supplement.* pp. S6-S9 (May 2004).

[11] Markel, M. 2007. *Technical Communication*, 8th ed., Boston, MA. Bedford/St. Martin's.

[12] Redish, J. 2007. *Letting Go of the Words: Writing Web Content that Works.* San Francisco, CA: Morgan Kaufmann Publishers.

[13] Clarke, S. 2003. *Using the Cognitive Dimensions, Continued - Learning Style 24 Nov 2003*. Available: http://blogs.msdn.com/stevencl/archive/2003/11/24/57079.aspx. [Accessed 2 February 2010].

[14] Rouet, J. F. 2006. *The Skills of Document Use: From Text Comprehension to Web-Based Learning*, Mahweh, NJ: Lawrence Erlbaum Associates.

[15] Wright, P. 1983. Manual dexterity: A user-oriented approach to creating computer documentation. In *Proc. ACM CHI '83 Conf.* (Boston, MA, 1983).

[16] Wright, P. 1991. Designing and evaluating documentation for I.T. users. In *Human Factors for Informatics Usability*. Shackel, B. and Richardson S., Eds., Cambridge, Cambridge University Press, pp. 343-358.

[17] Nielsen, J. 1993. *Usability Engineering*, Boston, MA: Academic Press.

[18] Bowles, T. M., Hensley M. K. and Hinchliffe, L. J. 2010. Best practices for online video tutorials: A study of student preferences and understanding. *Communications in Information Literacy.* vol. 4, no. 1, pp. 17-28 (2010).

[19] DeVaney, T. A. 2009. *Impact of Video Tutorials in an Online Educational Statistics Courses,* Dec. 2009. [Online]. Available: http://jolt.merlot.org/vol5no4/devaney_1209.htm. [Accessed 30 May 2012].

[20] Bridge, P. D., Jackson, M. and Robinson, L. 2009. The effectiveness of streaming video on medical student learning: A case study. *Wayne State University School of Medicine.* 19 August 2009. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2779626/pdf/MEO-14-RES00311.pdf. [Accessed 30 May 2012].

[21] W3Schools. 2012. *HTML5 <audio> Tag*. [Online]. Available: http://www.w3schools.com/html5/tag_audio.asp. [Accessed 30 May 2012].

## 9. ABOUT THE AUTHOR

Robert Watson developed software professionally for 17 years. After writing software for many products and designing many APIs, he stopped writing software and started writing about software as a programming writer at Microsoft. As a programming writer, he has been writing about software for software developers for almost 10 years. He holds a Master of Science degree in Human Centered Design and Engineering from the University of Washington with a focus on user-centered design and is currently studying for a PhD in Human Centered Design & Engineering and Global Technical & Communication Management.